

Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas
			6

Organización del Computador 2

Primer parcial – 07/05/2019

CORRIGIO:
ESTOY EN

1 (40) 40	2 (40) 40	3 (20) 20	100
--------------	--------------	--------------	-----

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

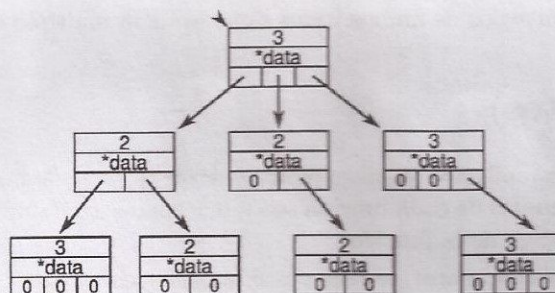
Ej. 1. (40 puntos)

Considerar un árbol construido utilizando para los nodos las siguientes dos estructuras:

```
typedef struct s_nodoDos {
    char tipo;
    void* data;
    void* derecha;
    void* izquierda;
} nodoDos;
```

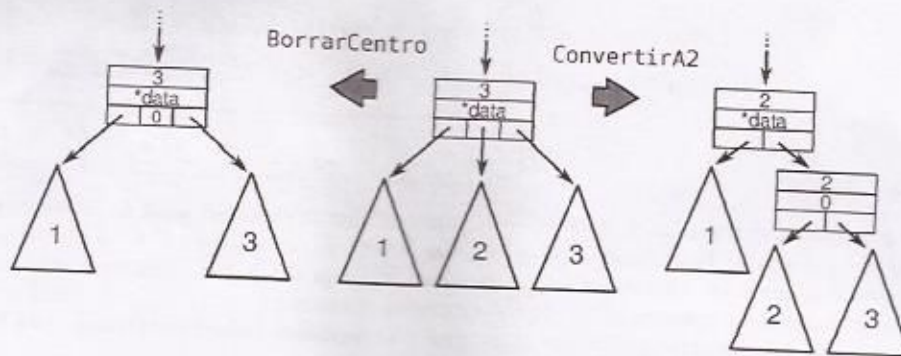
```
typedef struct s_nodoTres {
    char tipo;
    void* data;
    void* derecha;
    void* izquierda;
    void* centro;
} nodoTres;
```

Ejemplo:



El árbol puede tener nodos de cualquiera de los dos tipos, para identificar si el nodo almacena dos o tres punteros, se utiliza el primer byte de la estructura que indica con un 2 o 3 de que tipo es respectivamente.

- (20p) a. Implementar la función `void BorrarCentro(void **nodo, funBorrar *fb)`, que toma un doble puntero a un nodo, recorre el árbol y por cada nodo de tipo 3, borra todos los nodos del subárbol apuntado desde el puntero `centro`. Además, utiliza la función `fb` para borrar todos los datos que son eliminados.
- (20p) b. Implementar la función `void ConvertirA2(void **nodo)`, que toma un doble puntero a un nodo, recorre el árbol y por cada nodo de tipo 3 realiza la siguiente acción: Convierte al nodo de tipo 3 a tipo 2 conservando el dato original y el subárbol izquierdo. Además, como hijo derecho agrega un nuevo de tipo 2 cuyos subárboles izquierdo y derecho serán los subárboles central y derecho, respectivamente, del nodo original reemplazado. El dato del nuevo nodo debe ser un puntero a `null`.



Ej. 2. (40 puntos)

- (20p) a. Implementar la función `double* sumasRestas(float *A, short n)` que toma un arreglo de n números en punto flotante de simple precisión, realiza la operación a continuación y almacena el resultado en el mismo arreglo pero como punto flotante de doble precisión.

$$dst[i] = \frac{src[2 \cdot i] + src[2 \cdot i + 1]}{src[2 \cdot i + 1] - src[2 \cdot i]} \quad \text{con } i \text{ entre } 0 \text{ y } (n-1)/2$$

- (20p) b. Implementar la función `void remplazarExponentes(float *A, short n, uint8_t *exp)` que dado un arreglo de n números en punto flotante de simple precisión y un arreglo de n bytes, reemplaza los exponentes de cada uno de los valores de punto flotante por los almacenados en el arreglo de bytes.



Nota: considerar que los arreglos de ambos items tiene tamaño múltiplo de 16.

Ej. 3. (20 puntos)

Un nuevo entorno de compilación y linkeo permite llevar registro de la cantidad de veces que una función es llamada. Los contadores de cada función son almacenados como un número de 64bits exactamente antes del comienzo del código de la función.

Para incrementar estos contadores, se reemplazó la instrucción `ret` en las funciones por `jmp newRet`. Esta última es la encargada de incrementar el contador de la función y retornar de la misma a la función desde donde fue llamada.

- (15p) a. Implementar la función `newRet`. Considerar que todas las funciones son llamadas por medio de la instrucción `call IMM`. La codificación de la misma es:

...	1	2	...	9	...
...	opcode	dirección de la función			...

- (5p) b. Si una función pudiera ser llamada utilizando diferentes codificaciones de la instrucción `call`. ¿Es posible implementar la función `newRet`?

1) a) Implementar los fincos algunos bancos exitos

```
% Define OFFSET_TIPD 0
% Define OFFSET_DATA 8
% Define OFFSET_DER 16
% Define OFFSET_I24 24
% Define OFFSET_CEN 32
% Define SIZE_TIPD 32
```

6 HOJAS

bonosAkel: , RSI = *x1017 RSI = bn bnosAkel

```
PUSH RBP
MOV RBP, RSP
MOV RB, [RDI]
CMP RB, 0
JE .ortolVolo
MOV CL, [RB]
CMP CL, 2
JE .ortolde2
JMP .ortolde3
```

• .ortolVolo:

```
POP RBP
RET
```

• .ortolde2:

```
CALL bonosde2
POP RBP
RET
```

• .ortolde3

```
CALL bonosde3
POP RBP
RET
```


fonction 2 :

```
MOV
SUB RSP, 8
PUSH R12
PUSH R13

MOV R12, [RDI]
MOV R13, [RSI]

MOV RDI, [R12 + OFFSET_DATA]
CALL R13

LEA RDI, [R12 + OFFSET_I2R]
MOV RSI, R13
CALL fonction2

LEA RDI, [R12 + OFFSET_DER]
MOV RSI, R13
CALL fonction2

MOV RDI, R12
CALL FREE

POP R13
POP R12
ADD RSP, 8
RET
```

fonction 3

```
SUB RSP, 8
PUSH R12
PUSH R13

MOV R12, [RDI]
MOV R13, [RSI]
MOV RDI, [R12 + OFFSET_DATA]
CALL R13

LEA RDI, [R12 + OFFSET_I2R]
MOV RSI, R13
CALL fonction2

LEA RDI, [R12 + OFFSET_DER]
MOV RSI, R13
CALL fonction2

LEA RDI, [R12 + OFFSET_CEN]
MOV RSI, R13
CALL fonction2

MOV RDI, R12
CALL FREE

POP R13
POP R12
ADD RSP, 8
RET
```

kernel.Centho:

PUSH RBP
MOV RBP, RSP

PUSH R12
PUSH R13

MOV R12, [RDI]
MOV R13, RSI

CMP R12, 0
JE .ordelVocis

MOV CL, [R12+OFFSET_TIP0]
CMP CL, 2 ; is de tip ?
JE .again

LEA RDI, [R12+OFFSET_CEN]
CALL kernelAnch

• again:

LEA RDI, [R12+OFFSET_I2A]

MOV RSI, R13

CALL kernelCentho

LEA RDI, [R12+OFFSET_DER]

MOV RSI, R13

CALL kernelCentho

POP R13
POP R12
POP RBP
RET

• ordel vocis:

POP R13
POP R12
POP RBP
RET

comparativa2:

```
PUSH R12
MOV R12, RDI
CALL comparativa2
MOV R12, [R12]
CMP R12, 0
JE .noelvalor
LEA RDI, [R12 + OFFSET_0ER]
CALL comparativa2
LEA RDI, [R12 + OFFSET_12Q]
CALL comparativa2
POP R12
RET
```

comparativa2: ; RDI = xx valor

```
PUSH RBP
MOV RBP, RSP
PUSH R12
PUSH R13

MOV R12, RDI
MOV R13, [R12]
CMP R13, 0
JE .noelvalor

MOV CL, [R13 + OFFSET_TIPO]
CMP CL, 2
JE .noelvalor ; es equivalente a que haya un valor no 0

MOV RDI, SIZE_TPO2
CALL MALLOC
MOV CL, 2
MOV [RAX + OFFSET_TIPO], CL
MOV RDI, [R13 + OFFSET_CEN]
MOV RAX [RAX + OFFSET_12Q], RDI
```

```

MOV RDI, [R13+OFFSET_DER]
MOV RAX [RAX+OFFSET_DER], RDI

```

~~no se considera~~

```

MOV QWORD [RAX+OFFSET_DATA], 0

```

; ahora se coloca este como el hijo derecho del menu, pero se le da la buena
; direccion

```

MOV [R13+OFFSET_DER], RAX

```

```

MOV RDI, R12 ; llamas a llegar a esto parte A

```

```

CALL auxilior

```

- ~~entonces~~ ; llamas al llegar a esto parte B

```

POP R13

```

```

POP R12

```

```

POP RBP

```

```

RET

```

auxilior:

```

PUSH R12

```

```

MOV R12, RDI

```

~~CALL~~

```

MOV RDI, SIZE_TIPO_2

```

```

CALL MALLOC

```

```

MOV RDI, [R12]

```

```

MOV [R12], RAX

```

~~MOV RAX, [RDI+OFFSET_DATA]~~

```

MOV CL, 2

```

```

MOV [RAX], CL

```

```

MOV RSI, [RDI+OFFSET_DATA]

```

```

MOV [RAX+OFFSET_DATA], RSI

```

```

MOV RSI, [RDI+OFFSET_IZQ]

```

```

MOV [RAX+OFFSET_IZQ], RSI

```

```

MOV RSI, [RDI+OFFSET_DER]

```

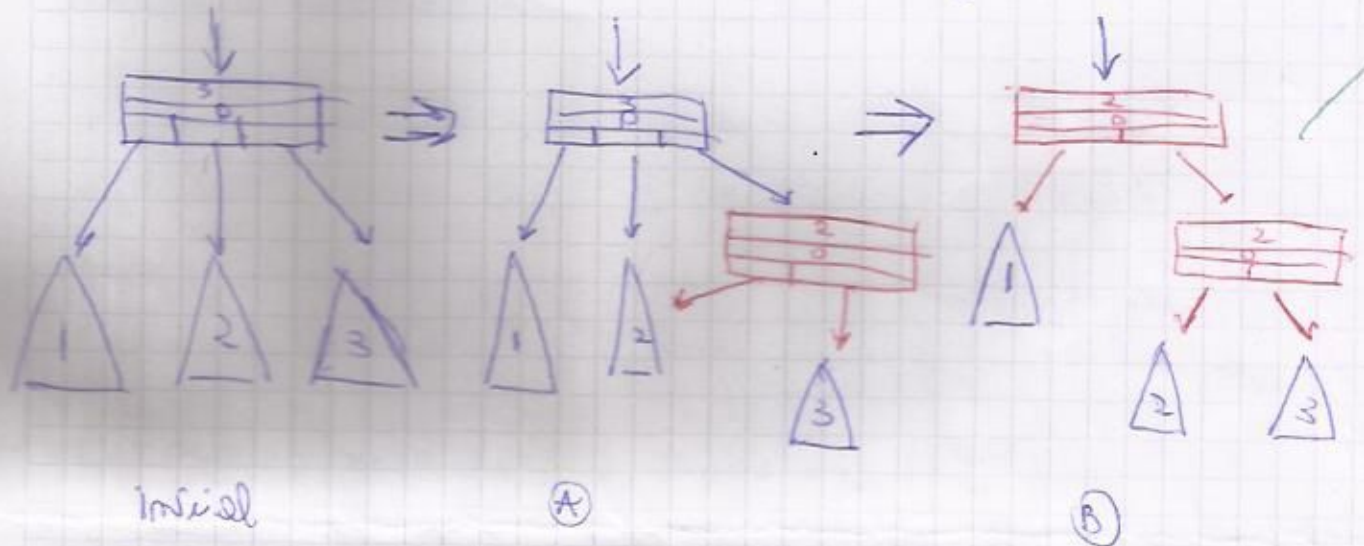
```

MOV [RAX+DER+OFFSET_DER], RSI

```


~~CALL~~
 CALL FREE
 POP R12
 RET

El procedimiento para convertirlo es puede ser gráfico como



Nota: En algunas funciones se ve como el stackframe se genera con
 agujeros o por los puntos ya quedados de las líneas

2. a)

MOV RAX, RDI

- ciclo:

CMP SI, 0

JE .fin

MOVDQU XMM0, [RDI] ; $XMM0 = |F_3|F_2|F_1|F_0|$

~~MOVXMM0, XMM0~~ ~~XMM0 = 0~~

MOVSLDUP XMM1, XMM0 ; $XMM1 = |F_2|F_2|F_0|F_0|$

MOVSHDUP YMM0, XMM0 ; $YMM0 = |F_3|F_3|F_1|F_1|$

PSRLQ XMM0, 4 ; $XMM0 = |0|F_3|F_2|F_0|$

PSRLQ XMM1, 4 ; $XMM1 = |0|F_3|F_3|F_1|$

CVTPS2PD XMM0, YMM0 ; $XMM0 = |d_2|d_0|$

CVTPS2PD XMM1, YMM1 ; $XMM1 = |d_3|d_1|$

MOVDQU XMM2, XMM1 ; $XMM2 = |d_3|d_1|$

ADDPD XMM1, XMM0 ; $XMM1 = |d_3+d_2|d_1+d_0|$

SUBPD XMM2, XMM0 ; $XMM2 = |d_3-d_2|d_1-d_0|$

DIVPD XMM1, XMM2 ; $XMM1 = \left| \frac{d_3+d_2}{d_3-d_2} \right| \left| \frac{d_1+d_0}{d_1-d_0} \right|$ ✓

MOVDQU [RDI], XMM1

ADD RDI, 16 ; aumento el puntero 16 bytes

SUB SI, 4 ; queda con 4 números

JMP .ciclo

~~REP .fin~~

RET ✓

b) SECTION DATA

konvention: $\downarrow b$ ^{FF} $0x80$, $0xFF$, $0xFF$, $0x80$

```

replacer_exp: db 0x00, 0xFF, 0xFF, 0xFF
               0x01, 0xFF, 0xFF, 0xFF
               0x02, 0xFF, 0xFF, 0xFF
               0x03, 0xFF, 0xFF, 0xFF

```

SECTION TEXT

MOVQQU xmm1, [repaches-4px]

MOVSS XMM2, [konsole]

PSHUFQ XMM2, 0 ; re base broadcast

◦ SiO_2
CMP SiO_2

TE. 43

```
MOVDQU XMM0, [RDI]
```

PAND χ_{MMO} , χ_{MMZ}

MOV SS, XMM3, [RCX]

PSHUFB XMM3, XMM1

PSLD XMM3, 23

~~PAID~~^{OR} XMM0, XMM3

MOVQ [RDI], xmm0

ADD R01, 16

```
ADD RCX, 4
```

308 51,4

JMP - Cila

RET

③

af El núcleo del problema es este: ¿Cómo obtener la dirección de mi función a partir de la dirección de regreso?

Una vez que se tiene esa información se puede modificar el contenido para ir a mi función.

Ahora, cuando se hace un RET el IP va a la dirección ~~inmediata~~ inmediata después del call a mi función, por lo que hay que leer la información inmediatamente antes a ~~esa~~ la dirección donde se llama.

Suponiendo que se necesita la dirección de C (se puede usar RB y RQ con mayor precisión)

newRet:

newRet:

MOV RB, [RSP] ; RB es ahora la dirección de regreso

SUB RB, 8 ; RB es ahora la dirección donde se halla
; la dirección de mi función

MOV RB, [RB] ; RB es ahora la dir de mi función

SUB RB, 8 ; RB es ahora un número a ~~la~~ donde se
; guarda la dirección de regreso

MOV RQ, [RB] ; conseguimos la dirección y guardamos

INC RQ

MOV [RB], RQ

RET

; notamos como se hizo manualmente

b) Si. Hacer que lean ~~la~~ la instrucción completa y reportar
en como han hallado donde se encuentra la dirección de mi función.

Si sólo me da codificación respecto al mismo número

OPCODE DIR entonces ~~el~~ lo hecho en el punto a) función

automáticamente

Y SI NO TIENE LA
DIRECCION?